Application of graphics processing units to search pipelines for gravitational waves from coalescing binaries of compact objects

2010 Class. Quantum Grav. 27 135009

(http://iopscience.iop.org/0264-9381/27/13/135009)

Class. Quantum Grav. **27** (2010) 135009 (13pp) doi:10.1088/0264-9381/27/13/135009

Application of graphics processing units to search pipelines for gravitational waves from coalescing binaries of compact objects

Shin Kee Chung[1], Linqing Wen[1,4], David Blair[1], Kipp Cannon[2] and Amitava Datta[3]

[1] School of Physics, The University of Western Australia, 35 Stirling Highway, Crawley WA 6009, Australia
[2] Division of Physics, Mathematics and Astronomy, California Institute of Technology, 1200 California Blvd., Pasadena, CA 91125, USA
[3] School of Computer Science and Software Engineering, The University of Western Australia, 35 Stirling Highway, Crawley WA 6009, Australia

E-mail: lwen@cyllene.uwa.edu.au

Received 17 January 2010, in final form 24 March 2010
Published 14 May 2010
Online at stacks.iop.org/CQG/27/135009

Abstract
We report a novel application of a graphics processing unit (GPU) for the purpose of accelerating the search pipelines for gravitational waves from coalescing binaries of compact objects. A speed-up of 16-fold in total has been achieved with an NVIDIA GeForce 8800 Ultra GPU card compared with one core of a 2.5 GHz Intel Q9300 central processing unit (CPU). We show that substantial improvements are possible and discuss the reduction in CPU count required for the detection of inspiral sources afforded by the use of GPUs.

PACS numbers: 04.25.Nx, 04.30.Db, 04.80.Nn, 95.55.Ym

(Some figures in this article are in colour only in the electronic version)

1. Introduction

It is an exciting time for gravitational-wave astronomy. Several ground-based gravitational-wave (GW) detectors have reached (or approached) their design sensitivity, and are coordinating to operate as a global array. These include the three LIGO detectors in Louisiana and Washington states of the USA, and the Virgo detector in Italy. The LIGO detectors have already completed their ground-breaking fifth science run. An integrated full year's worth of science data summing up to more than 10 terabytes has been accumulated from all three interferometers in coincidence. Enhanced LIGO started to operate in June 2009

[4] Also, International Centre for Radio Astronomy Research, UWA, 35 Stirling Hwy, Crawley, WA 6009, Australia.

and plan to continue until the end of 2011 with similar sensitivity [1]. Starting in 2011, an upgrade of Enhanced LIGO to Advanced LIGO (expected to operate in 2014) will enable a tenfold improvement in sensitivity, allowing the detectors to monitor a volume of the universe 1000 times larger than the current detectors. The detection rate of signals from coalescing binaries of compact objects for these advanced detectors is estimated to be tens to hundreds of events per year [1]. The detection of the first GW is virtually assured with Advanced LIGO.

1.1. Search for gravitational waves from coalescing compact binaries

Coalescing binaries of compact objects consisting of neutron stars and black holes are among the most important GW sources targeted by the current large-scale GW detectors [2, 3] as these sources produce a very distinct pattern of gravitational wave. The optimal way to detect the known waveforms in noisy data is to perform a matched filtering. The matched filtering technique is performed by calculating the correlation between the GW data and a set of known or predicted waveform templates [4, 5]. The post-Newtonian expansion method is used to approximate the nonlinear equations that describe the motion of coalescing binaries and wave generation in the creation of waveform templates [6]. For spinless, circular, binary systems each waveform is specified by a set of parameters including a pair of individual masses $I = (m_1, m_2)$, constant orbital phase offset at formal coalescence α and effective distance D_{eff} from the detector. In our tests, second-order post-Newtonian orbital phases and Newtonian amplitude were used.

The application of the matched filtering technique to on-going searches for gravitational waves from coalescing binaries of compact objects is described in [2] and is summarized as follows. The template waveforms corresponding to $\alpha = 0$ and $\alpha = \pi/2$ form an orthonormal set [7]. For a given mass pair $I = (m_1, m_2)$, the waveforms with $\alpha = 0$ and $\alpha = \pi/2$, denoted by h_c^I and h_s^I, are approximately related by $\tilde{h}_c^I(f) = -\mathrm{i}\tilde{h}_s^I(f)$ where $\tilde{h}$ is the Fourier transform of h. Exploiting this, the matched-filter output $z(t)$ is a complex time series defined as

$$z(t) = x(t) + \mathrm{i}y(t) = 4\int_0^\infty \frac{\tilde{h}_c^I(f)\tilde{s}^*(f)}{S_n(f)}\,\mathrm{e}^{2\pi\mathrm{i}ft}\,\mathrm{d}f \tag{1}$$

where $S_n(f)$ is the one-sided strain noise power spectral density, $\tilde{s}^*(f)$ is the Fourier transform complex conjugate of the detector's calibrated strain data $s(t)$, $x(t)$ is the matched-filter response of h_c^I and $y(t)$ is the matched-filter response of h_s^I. For initial LIGO detectors, $S_n(f)$ is defined as

$$S_n(f) = \left(\frac{4.49f}{f_0}\right)^{-56} + 0.16\left(\frac{f}{f_0}\right)^{-4.52} + 0.52 + 0.32\left(\frac{f}{f_0}\right)^2 \tag{2}$$

where $f_0 = 150\,\text{Hz}$ [8]. In practice, the noise power spectral density from the Science Requirement Document is used [9]. Maximizing over the coalescence phase α, the signal-to-noise ratio (SNR) $\rho(t)$ is the absolute value of the scalar product between a normalized template and the detector output in the frequency domain [2]:

$$\rho(t) = \frac{|z(t)|}{\sigma} \tag{3}$$

where the normalization factor σ is calculated from the variance

$$\sigma^2 = 4\int_0^\infty \frac{\left|\tilde{h}_c^I(f)\right|^2}{S_n(f)}\,\mathrm{d}f. \tag{4}$$

For stationary and Gaussian noise, this ρ is the optimal detection statistics for a single detector.

Table 1. Mass ranges and their corresponding number of templates obtained from running the actual template generating program in the LIGO searching pipeline valid for LIGO Hanford detector H1.

Mass range (solar masses)	Number of templates
10.0–11	7
9.0–11	15
8.0–11	26
7.0–11	48
6.0–11	85
5.0–11	163
4.0–11	317
3.0–11	718
2.0–11	2111
1.6–11	3734
1.4–11	5222

The number of templates needed depends on the parameter volume needed to be searched. The two masses of the compact binary objects were used as the main parameters in our experiment as we were focusing on spinless and circular binaries of compact objects. The low frequency cutoff was set to be 40 Hz while a high frequency cutoff is the Nyquist frequency (half of the sampling frequency, 2 kHz in our case) or the frequency of the innermost stable circular orbit (ISCO, e.g. [10]) for the analyzed template, whichever is lower. In our experiments, the mass ranges were varied and the number of templates corresponding for each mass range was calculated. The mass ranges and their corresponding number of templates are listed in table 1. In order to achieve $< 5\%$ mismatch (i.e., $1 - \rho/\rho_{\mathrm{exp}} < 5\%$ where ρ_{exp} is the expected SNR when the template waveform exactly matches the signal in the data), thousands of templates are required [4] to analyze a data segment for mass ranges of 1.4–11 solar masses for each individual member of the binary. In the currently running search pipeline described in [2], each data segment is made up of 256 s of detector data down-sampled to 4096 Hz giving 2^{20} data points. This means that thousands of FFTs, each of approximately 1 million data points, are required to filter one data segment through the template bank.

1.2. The χ^2 consistency test

In order to verify the signals and reject non-Gaussian transient noise, the χ^2 consistency test [11] is used as a time–frequency veto. The integral in equation (1) is split into p frequency bands such that each contributes an equal amount to the SNR, and this yields p time series, $z_l(t)$, where l ranges from 1 to p. In stationary Gaussian noise with or without a GW signal, the statistic [2]

$$\chi^2(t) = \frac{p}{\sigma^2} \sum_{l=1}^{p} |z_l(t) - z(t)/p|^2 \tag{5}$$

is a χ^2-distributed random variable with $\nu = 2p - 2$ degrees of freedom. Transient departures from Gaussian noise that are poor matche for GW templates, or 'glitches', are associated with large values of the χ^2 statistic, and this can be used to reject such noise events [2].

1.3. Motivation of GPU acceleration

As the above discussion implies, much computing power is required to search for GWs from these sources. With current technology, more than 50 CPU cores are required to finish the detection phase of the analysis within the duration of the data. The χ^2 waveform consistency test requires another 50 CPU core processing power in order to complete the analysis in real time. Furthermore, determination of sky directions of these sources requires hours to days of CPU core time. The long time scales required for detection, verification and localization pose a serious problem for prompt follow-up observations of these sources by optical telescopes. Such follow-up observations are expected not only to provide a firm proof that a gravitational wave has been detected but also to provide insight into the physics associated with the events. Much faster processing is therefore required for real-time detection and determination of source directions.

In this paper, we propose a cost-effective and user-friendly alternative to reduce the computational cost in GW searches by using the graphics processing unit (GPU) to accelerate the data processing. The on-going searches for GW signals from detector data are ideal for these massively parallel processors. This is due to the fact that the same algorithm is applied to different data segments independently and repeatedly and that the latency of transferring data between the GPU and the CPU is negligible compared to analysis time. We report here the result of a first test, using a GPU in a modified existing data analysis pipeline described in [2] to search for GW signals from coalescing binaries of compact objects (denoted the *inspiral search pipeline*). A previous report can be found in [12].

2. Graphics processing units and CUDA

Graphics processing units (GPUs) were originally designed to render detailed real-time visual effects in computer games. The demands for GPUs in the gaming industry have enabled GPUs to become low-cost but very efficient computing devices. Due to the nature of its hardware architecture, it is advantageous to use GPUs for solving parallel problems that fit the single-instruction-multiple-data (SIMD) model. While the capability of GPUs in high performance computing has been recognized since 2005 [13], general purpose GPU (GPGPU) parallel computing has really become viable only recently. This is due to the release of the C-programming interface CUDA (compute unified device architecture) by NVIDIA Corporationr in February 2007 [14]. The introduction of CUDA enabled scientists in a much broader community to program on GPUs by calling C-libraries. Previously, one would have to translate a general problem into graphical pixel models in order to make use of the GPUs. Remarkable speed-ups of up to a factor of 100 have been reported in many applications including those of important astronomical applications [13, 15]. A sizable CUDA library is now available for basic scientific computations. This includes linear algebraic computation, FFT and tools for Monte Carlo simulation. The use of GPGPU techniques as an alternative to distributed computer clusters has also become a real possibility.

One successful application of GPU acceleration was in the computation of molecular dynamics. Anderson, Lorenz and Travesset [16] implemented CUDA algorithms to handle the core calculations for molecular dynamics. Anderson *et al* in fact slightly altered one of the core algorithms of molecular dynamics for CUDA so that some of the calculations will be repeated instead of accessing the same information from memory. This avoided the problem of inefficiency of CUDA in accessing random memory locations. The CUDA program running in a system with one single GPU and one single CPU was found to be performing at equivalent

level of a fast computer cluster with 36 cores. Such a cluster consumes more power compared to a single computer with a GPU [16].

There exist several studies of CUDA implementations in the field of astronomy. Belleman *et al* [17] studied the CUDA implementations of *N*-body simulations, following the studies of Zwart *et al* [13] who used GPUs in *N*-body simulations before the release of CUDA. The CUDA implementation of *N*-body simulations developed by Belleman *et al* was able to achieve up to 100 times speed-up compared to that of CPU. Harris *et al* [15] conducted the CUDA implementations for calculating the signal convolutions for a radio astronomy array. In this application, signals from each antenna are combined using convolutions. This allows an array of antennas to be used to achieve high angular resolution. The CUDA implementation of this process showed two orders of magnitude speed-up. The use of GPUs in GW data analysis has not been reported before writing this (see [18] for an earlier proposal).

2.1. Crucial elements regarding GPU acceleration

Programming for the GPU with CUDA is different from general purpose programming on the CPU due to the extremely multi-threaded nature of the device. For an algorithm to execute efficiently on the GPU, it must be cast into a data-parallel form with many thousands of independent threads of execution running the same lines of code, but on different data. Because of this simultaneous execution, one thread cannot depend upon the output of another, which can pose a serious challenge when trying to cast some algorithms into a data-parallel implementation.

Specifically, in CUDA, these independent threads are organized into blocks which can contain anywhere from 32 to 512 threads each, but all blocks must have the same number of threads. Each block executes identical lines of code and is given an index to identify which piece of the data it is to operate on. Within each block, threads are numbered to identify the location of the thread within the block. Any real hardware can only have a finite number of processing elements that operate in parallel. In particular, a single GeForce 8800 Ultra GPU that was used in our work contains 16 multiprocessors. A single multiprocessor can execute a number of blocks concurrently (up to resource limits) in warps of 32 threads.

CUDA programs are divided into two parts: host functions and kernel functions. The host functions are the code that run in the CPU and are able to invoke kernel functions. The kernel functions are the code that run in the GPU. These functions are automatically executed by the threads in each block of the GPU. CUDA programmers need to specify the number of blocks and the number of threads per block for the kernel functions. Threads in the same block can be synchronized (hold and wait until all threads have finished previous tasks) and communicate (access the data or output of other threads). However, threads in different blocks cannot be synchronized. This imposes a great restriction in CUDA programming. We need to carefully choose the number of blocks and threads to obtain the highest performance.

The memory structure of a GPU is organized in a convenient way. There is a global memory (768 MB for GeForce 8800 Ultra) that can implement read-and-write operations simultaneously. This hides the latency of data accessing between processors and memory. Meanwhile, each block of threads executing on a multiprocessor has access to a smaller but faster shared memory (16 KB for GeForce 8800 Ultra GPU). Proper use of the threads and memory access are therefore crucial for optimization of the performance of GPU-computing. It is also necessary to copy data between CPU and GPU. This introduces a latency that needs to be taken into account when optimizing the GPU-computing. Specifically, this means that the algorithms must be designed so that the computational time is much larger than the latency.

3. Application of GPUs to inspiral search pipeline

The search pipelines for GWs from coalescing compact binaries have been developed since 2000 [19]. The current pipeline [2] has been used for the past five successful science runs on real data from the LIGO detectors [20] and the source code is publically available in the LSC Algorithm Library (LAL) [21].

According to our experiments, the most time-consuming part (80% of the total run time) of the existing search pipeline is the forward Fourier transform and its inverse. Our implementation replaces the Fastest Fourier transform in the West (FFTW) [22] used by the existing pipeline with the CUDA fast Fourier transform [14], denoted as CUDA FFT in this paper. The CUDA FFT also provides functions that can calculate several FFTs in parallel, as in FFTW. We identified modules in the pipeline that perform FFTs in sequence and rewrote them using batched CUDA FFTs.

Our second task was to accelerate the χ^2 waveform consistency test described in section 1.2. This is by far the most computational-intensive module in the pipeline. Within the program, the most time-consuming part lies in a loop of FFTs that operate on different data segments in series, and a double loop that calculates the χ^2 statistics from the output of the FFTs. We took advantage of data parallelism by copying large segments of data into the global memory of the GPU card, and perform the χ^2 calculation in parallel on these data segments.

In summary, our applications of the GPUs on the inspiral search pipeline include the use of the existing CUDA FFTs for the SNR and χ^2 calculations, and the direct implementation of parallel computation for the χ^2 calculation. A stand-alone version of the inspiral search pipeline (that normally runs on computer clusters) was run on a single core of a Dell Inspiron 530 computer with a 2.5 GHz Intel Core 2 Quad 9300 CPU. The GPU card used for timing comparison is an NVIDIA GeForce 8800 Ultra installed in the same computer, using CUDA version 1.1. Stationary colored Gaussian noise with a spectrum matching the Initial LIGO Science Requirement Document [9] for Hanford detector H1 was used for the test. In all our tests and implementation, we purposely kept all relevant search parameters close to a real search as implemented in the past few science runs. Further acceleration could be expected once we consider flexible search parameters or rewrite a much larger fraction of the code.

3.1. Implementation of the CUDA Fourier-transform

As described in the previous subsection, the GW search pipeline spends the majority of the time in performing FFTs. LAL uses Fastest Fourier Transform in the West (FFTW) for performing FFTs. FFTW was developed by Frigo and Johnson for improving the performance of FFTs' calculations by CPUs [22]. The main feature of FFTW is that it uses a *planner* to learn the fastest way to perform FFT in a computer. This *planner* constructs plans for executing FFTs in a fast way, and the plans are re-used for each execution of the FFT in a particular computer [22]. Similarly, CUDA has a complete FFT library developed by NVIDIA which also uses a *planner* following the design of FFTW [14].

Our first experiment was to compare the performance of the CUDA FFT to FFTW. To do this, a program that generates data sets and performs CUDA FFTs on the data was developed. The transferring of data from the host CPU to the GPU (and vice versa) and the CUDA FFT calculation was put into a loop so that appropriate repetitions of the calculations could be timed. This means that the time measured is the data transfer time plus the CUDA FFT execution time. A similar program that uses FFTW instead of the CUDA FFT on the same set of data was also developed and timed. The program that uses FFTW does not need to transfer data from elsewhere.

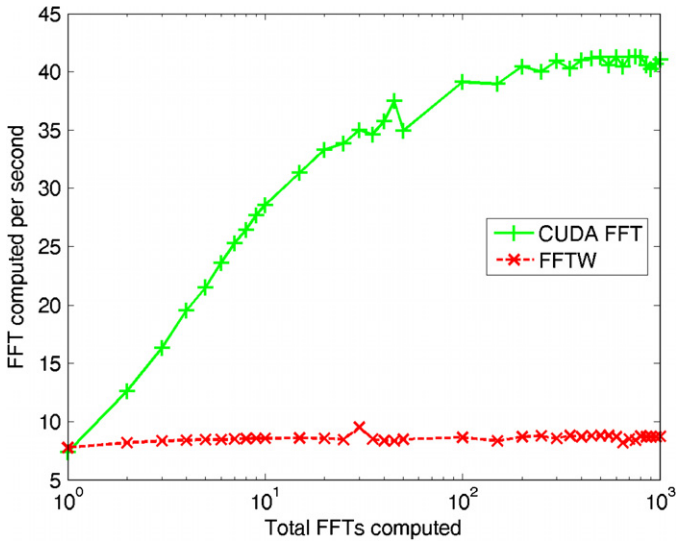

Figure 1. FFTs executed per second as a function of the total number of FFTs executed with 2^{20} data points each. The (green) solid line shows the number of FFTs executed by CUDA, while the (red) dashed line shows the number of FFTs executed by FFTW.

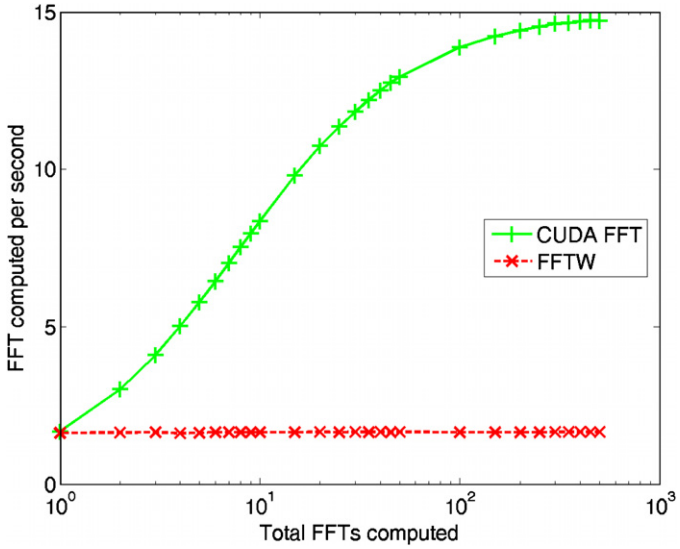

Figure 2. FFTs executed per second as a function of the total number of FFT executed with 4×2^{20} data points each. The (green) solid line shows the number of FFTs executed by CUDA, while the (red) dashed line shows the number of FFTs executed by FFTW.

The comparison of CUDA FFT and FFTW is shown in figures 1 and 2. The graphs show the number of FFTs executed per second (vertical axis) against the number of FFTs executed (horizontal axis). The vertical axis therefore indicates the capability of the hardware. Higher values on the vertical axis indicate better capability of the computer in performing FFT operations. The performance of the CUDA FFT shows a fast rise at a low number of FFTs

and reaches a plateau at larger numbers. The number of FFTs was incremented by one for each step at the beginning to show clearly the initial quick rise in performance. As the graph flattens at higher number of FFTs, the increment is 50 for each step. In the experiment, the time for transferring data from the host CPU memory to the GPU memory was found to be negligible compared to the CUDA FFT execution time.

An interesting feature shown in figures 1 and 2 is the poor performance of the CUDA FFT for small numbers of FFTs. As the experiments were carried out, we found that there is an initialization period of 'warming up' time associated with the GPU before any executions could be performed on it (even before we can start transferring data to the GPU). Our GPU was tested in CUDA version 1.1 with a program that repeatedly allocates memory in the GPU. We found that there is generally a huge delay in allocating the first memory space, in the order of 100 ms. The next memory allocation takes less than 1 ms. Therefore, the GPU does not perform well when the number of FFTs performed is small where the initialization period is a significant fraction of the total time. In fact, if we perform only one FFT, then the CUDA FFT is slower than FFTW. However, there is a quick rise in the performance of the CUDA FFT at the beginning when the number of FFTs is increased. Both the CUDA FFT performance curves flatten out at about 1000 total FFTs when the 'warming up' time is much smaller than the total execution time.

Figure 1 shows that CUDA FFT can execute about 40 FFTs per second for 2^{20} data points, while FFTW can execute about 8 FFTs per second. That means CUDA FFT is performing five times faster than FFTW. Figure 2 shows about 7.5 times speed-up for 4×2^{20} data points. In the inspiral search pipeline, the FFTs are performed on 2^{20} data points. Our results indicate that more speed-up can be achieved if more data points are used. Overall, it is only worth the effort to use CUDA FFTs if a large number of FFTs are to be executed.

We developed an interface of the CUDA FFT to be used by the GW search community in general. This was done by adding the interface into LAL for calling the CUDA FFT library which can be manually activated or deactivated by LAL users. This is the first time that a GPU interface was successfully developed for LAL and tested with real applications.

3.2. Acceleration using data parallelism of GPUs

We applied the GPU data parallelism to the most computationally intensive and time-consuming stage of the GW search pipeline, the χ^2 test. The χ^2 test splits the inspiral template into 16 pieces in the frequency domain, and convolves the Fourier transform of the input data with the split 16 time series representing the contribution to the template's net SNR (as a function of time) from each of its 16 pieces. Suitably normalized, the sum of the square magnitudes of these 16 time series is χ^2-distributed when the input data contain stationary Gaussian noise and a possibly absent GW signal, and testing for this forms the basis of a waveform consistency test.

The conversion of the χ^2 test to a GPU implementation was done in two parts. Firstly, 16 sequential inverse FFTs were replaced with 16 parallel inverse FFTs. This part was implemented by calling existing CUDA functions from the host code. The comparison of this implementation to the original one is shown in figure 3. Secondly, we implemented the GPU data parallelism on the χ^2 test described in section 1.2. Table 2 shows the χ^2 implementation in C and the GPU-accelerated implementation. $a_{i,l}$ and $b_{i,l}$ represent the real and imaginary parts of $z_l(t) - z(t)/p$ in equation (5) respectively. N is the number of data points being analyzed, i ranges from 0 to $N - 1$ and p is the total number of frequency bands. In the original implementation, a double loop was used to calculate χ^2 values sequentially with $p = 16$ and $N = 2^{20}$. A total of 16×2^{20} χ^2 values were thus calculated independently. For

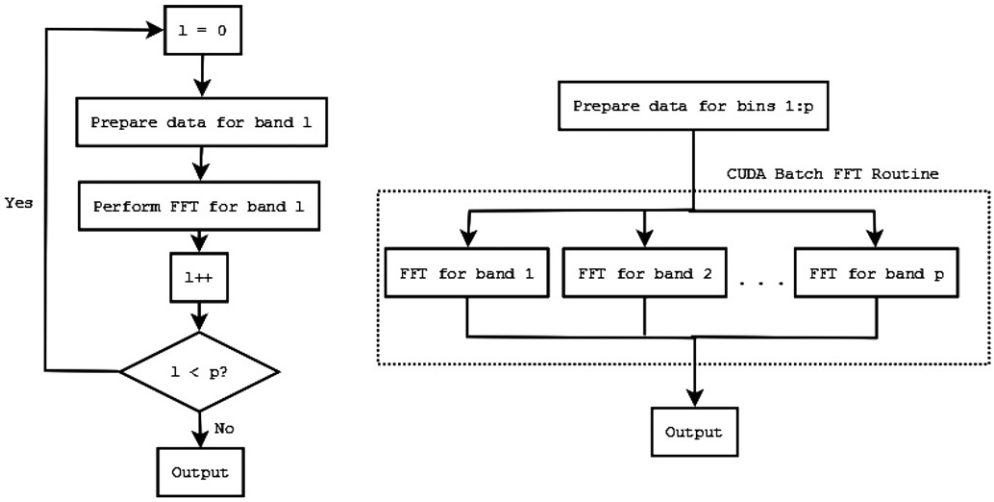

Figure 3. The comparison flow chart of the sequential FFTs and batched FFTs execution in the χ^2 calculation.

Table 2. Algorithm comparison of χ^2 implementation in C and CUDA.

Original C implementation	CUDA implementation
`for(i = 0; i < N; i++){`	Thread i_1:
`for(l = 0; l < p; l++){`	`for(l = 0; l < p/2; l++)`
$\chi^2_i += a^2_{i,l} + b^2_{i,l}$	$\chi^2_{i_1} += a^2_{i,l} + b^2_{i,l}$;
}	
}	Thread i_2:
	`for(l = p/2; l < p; l++)`
	$\chi^2_{i_2} += a^2_{i,l} + b^2_{i,l}$;
	Synchronizing all threads:
	$\chi^2_i += \chi^2_{i_1} + \chi^2_{i_2}$;

each time t, values from all 16 frequency bands were then added (cf equation (5)), yielding a total of 2^{20} outputs. In the CUDA implementation, we replaced this double loop with a single loop in the parallel threads. This part was implemented in a custom kernel function where 4×2^{10} blocks of 2^9 threads each were used. Each thread calculates eight χ^2 values and sums them together sequentially. Adjacent threads were used to calculate χ^2 values of the same frequency band. The results from every two adjacent threads are then summed at the end of this single loop after a synchronization was executed. This approach was found to give optimal performance. The thread numbers are chosen to be multiples of the GPU warp size 32 (explained in section 2.1) and able to divide the loop number exactly.

3.3. Results

The timing results of our implementation of CUDA FFT and data parallelism for χ^2 implementation in the inspiral search pipeline are shown in figures 4–7. About 4x speed-up can be achieved by simply enabling this CUDA FFT interface for LAL (figures 4 and 5).

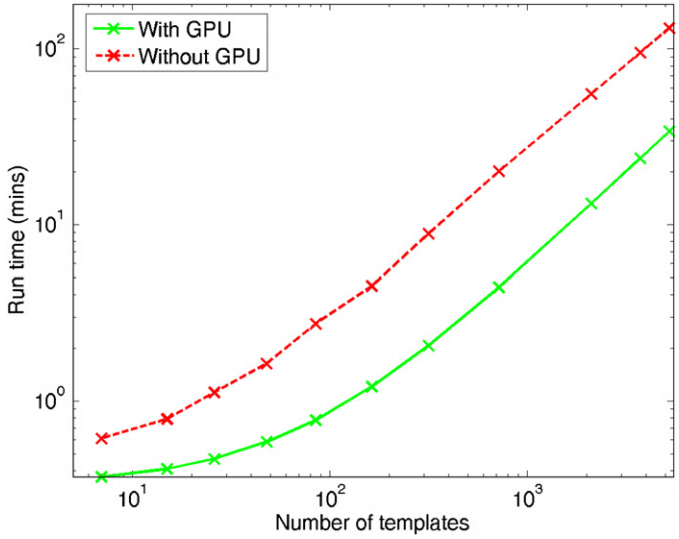

Figure 4. The run time of the inspiral searching pipeline without performing the χ^2 test. The (green) solid line shows the run time of inspiral search with GPU, while the (red) dashed line shows the run time of inspiral search without GPU.

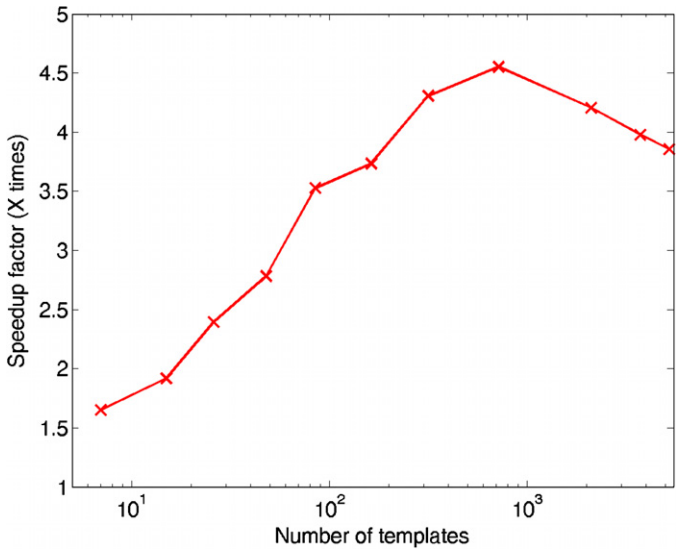

Figure 5. The speed-up factor, calculated as run time without GPU divided by run time with GPU.

The run time of the inspiral pipeline was shown in figure 4 and the speed-up factor was shown in figure 5.

In figure 6, the vertical axis shows the run time of the inspiral search pipeline while the horizontal axis shows the number of templates used for the search. It is shown that, at about 700 templates, the inspiral search using the original χ^2 implementation took about 6 h to complete, while it required only about 20 min to complete with our GPU implementation.

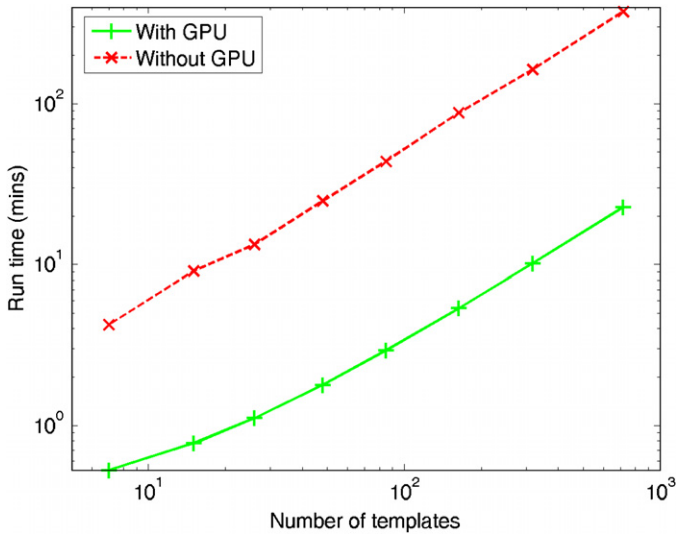

Figure 6. The run time for executing the inspiral search with χ^2 veto enabled, both with and without GPU acceleration. The (green) solid line shows the run time of inspiral search with the GPU, while the (red) dashed line shows the run time of inspiral search without the GPU.

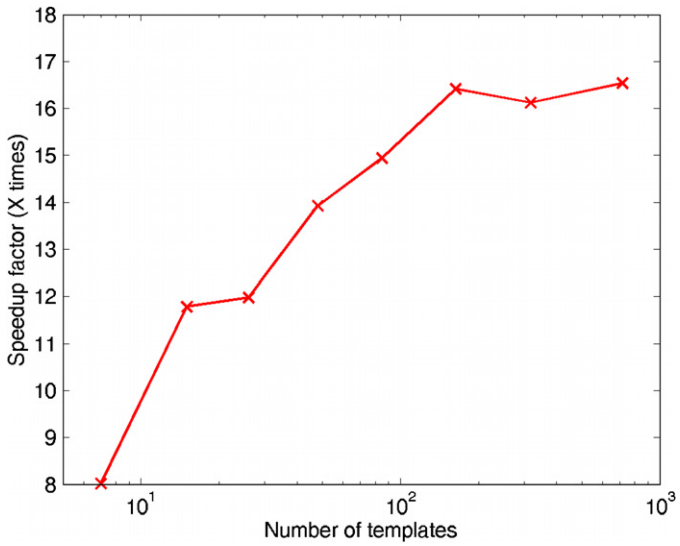

Figure 7. The speed-up factor (see the text in section 3.3) using the CUDA implementation of the χ^2 calculation.

The speed-up factor of the GPU implementation compared to the CPU-only implementation is shown in figure 7. The vertical axis shows the speed-up factor—the run time of the original CPU-only implementation divided by the run time of the GPU implementation. About 16 times speed-up was observed. This means that the number of computers needed to perform the analysis in the same amount of time can be significantly reduced. A normal

computer with integrated graphics should consume about 220 W of power, or 3520 W for 16 single core computers, or 880 W for 4 quad core computers. In comparison, a single computer with GeForce 8800 Ultra consumes about 340 W of power. We could save some hardware costs and also reduce power consumption by a significant amount.

An accuracy test was performed by calculating the fractional difference between the outputs produced by the new inspiral search with CUDA FFT including the parallel χ^2 implementation and the original inspiral search with FFTW in the mass range of 3.0–11.0 solar masses. About 5×10^4 events were identified from the data, each with measured SNR and χ^2 values. We found that more than 99% of the SNRs had less than 0.03% difference, while 99% of the χ^2 values had less than 0.5% difference.

4. Conclusion

We have shown that GPUs can significantly improve the speed of GW data analysis. A speed-up of fourfold to fivefold in the existing inspiral search pipeline can be achieved by simply enabling the CUDA FFT. Note that the CUDA FFT has already been introduced to LAL, meaning that other GW search pipelines can use it provided GPUs are available. We achieved a 16-fold speed-up in total by using a specially written parallel GPU implementation of the χ^2 test, a waveform consistency test used within the pipeline. We expect further speed-ups if we are allowed to change some of the search parameters. For instance, if we change the number of data points for FFTs from the currently used 1 million to 4 millions, another factor of 2 speed-up can be achieved. Also, further acceleration is expected if we replace more components in the pipeline with specially written GPU implementations.

Our experiments were performed using a single GPU, while current new personal computers can be equipped with more than three GPUs. We would expect more than 48-fold speed-up using a 3-GPU system when running a single-threaded search pipeline. Furthermore, if we can use the newest GPU on the market, which has about 1 TFLOPS of computing power, and assuming that the performance of these GPUs scales linearly, we would expect more than a 100-fold speed-up in a single core desktop computer.

Acknowledgments

We are grateful to Chad Hanna, Drew Keppel, Phil Ehrens, Stuart Anderson, Alan Weinstein, Yanbei Chen, Karen Haines, Shaun Hooper, Adam Mercer and Oliver Bock for discussions of this work. This work is supported in part by the David and Barbara Groce start-up fund at Caltech, the Caltech SURF program, the Alexander von Humboldt Foundation's Sofja Kovalevskaja Programme funded by the German Federal Ministry of Education and Research, the Australian Research Council and the WA Centres of Excellence Program.

References

[1] Smith J R 2009 (for the LIGO Scientific Collaboration) The path to the enhanced and advanced LIGO gravitational-wave detectors *Class. Quantum Grav.* **26** 114013
[2] Abbott B *et al* 2004 Analysis of LIGO data for gravitational waves from binary neutron stars *Phys. Rev.* D **69** 122001
[3] Hughes S A 2009 Gravitational waves from merging compact binaries arXiv:0903.4877
[4] Owen B J and Sathyaprakash B S 1999 Matched filtering of gravitational waves from inspiraling compact binaries: computational cost and template placement *Phys. Rev.* D **60** 022002
[5] Finn L S 1992 Detection, measurement, and gravitational radiation *Phys. Rev.* D **46** 5236

[6] Blanchet L, Iyer B R, Will C M and Wiseman A G 1996 Gravitational waveforms from inspiralling compact binaries to second-post-Newtonian order *Class. Quantum Grav.* **13** 575
[7] Dhurandhar S V and Sathyaprakash B S 1994 Choice of filters for the detection of gravitational waves from coalescing binaries: II. Detection in colored noise *Phys. Rev.* D **49** 1707
[8] Damour T, Iyer B R and Sathyaprakash B S 2001 Comparison of search templates for gravitational waves from binary inspiral *Phys. Rev.* D **63** 044023
[9] Lazzarini A and Weiss R 1996 LIGO Science Requirements Document (SRD), LIGO-E950018-02-E, available from: http://www.ligo.caltech.edu/docs/E/E950018-02.pdf
[10] Buonanno A 2002 Gravitational waves from inspiralling binary black holes *Class. Quantum Grav.* **19** 1267
[11] Allen B 2005 χ^2 time-frequency discriminator for gravitational wave detection *Phys. Rev.* D **71** 062001
[12] Chung S K 2008 *Honours Thesis* School of Computer Science and Software Engineering, The University of Western Australia
[13] Portegies Zwart S F, Belleman R G and Geldof P M 2007 High-performance direct gravitational N-body simulations on graphics processing units *New Astron.* **12** 641
[14] NVIDIA Corporation 2007 NVIDIA CUDA Programming Guide 1.1, available from: www.nvidia.com/object/cuda_develop.html
[15] Harris C, Haines K and Staveley-Smith L 2008 GPU accelerated radio astronomy signal convolution *Exp. Astron.* **22** 129
[16] Anderson J A, Lorenz C D and Travesset A 2008 General purpose molecular dynamics simulations fully implemented on graphics processing units *J. Comput. Phys.* **227** 5342
[17] Belleman R G, Bédorf J and Portegies Zwart S F 2008 High performance direct gravitational *N*-body simulations on graphics processing units: II: an implementation in CUDA *New Astron.* **13** 103
[18] Győző E 2008 Parallel computation on graphical processor units with an eye on gravitational-wave data analysis needs, available from: http://www.ligo.caltech.edu/docs/T/T070308-00.pdf
[19] Tagoshi H *et al* 2001 First search for gravitational waves from inspiraling compact binaries using TAMA300 data *Phys. Rev.* D **63** 062001
[20] Brown D A *et al* 2004 Searching for gravitational waves from binary inspirals with LIGO *Class. Quantum Grav.* **21** 1625
[21] Data Analysis Software Working Group, LAL Home Page, available from: https://www.lsc-group.phys.uwm.edu/daswg/projects/lal.html
[22] Frigo M and Johnson S G FFTW. MIT, available from: www.fftw.org